

PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau



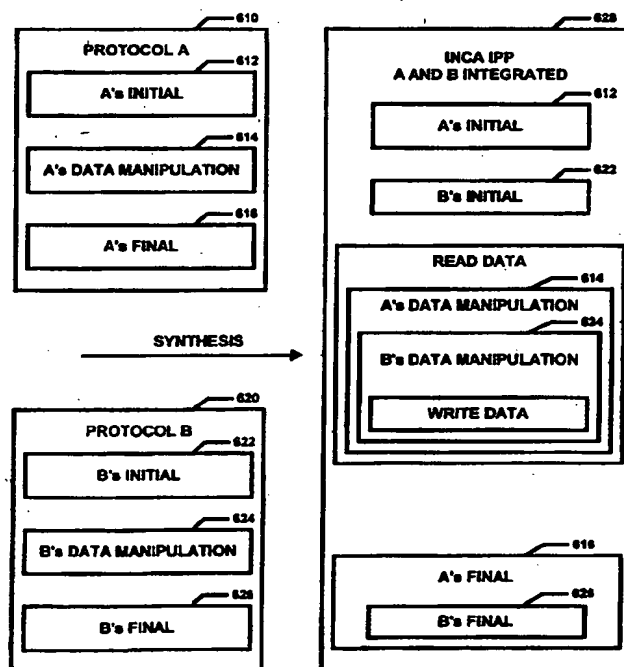
INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification ⁶ : H04L 12/00		A2	(11) International Publication Number: WO 99/26377
			(43) International Publication Date: 27 May 1999 (27.05.99)
(21) International Application Number: PCT/US98/24395			(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).
(22) International Filing Date: 16 November 1998 (16.11.98)			
(30) Priority Data: 08/972,157 17 November 1997 (17.11.97) US			
(71) Applicant: MCMZ TECHNOLOGY INNOVATIONS LLC [US/US]; 11029 Grassy Knoll Terrace, Germantown, MD 20876 (US).			
(72) Inventor: SCHUG, Klaus, H.; 3360 Gunnison Drive, Fort Collins, CO 80526-2790 (US).			
(74) Agents: ROBERTS, Jon, L. et al.; Roberts & Brownell, LLC, Suite 212, 8381 Old Courthouse Road, Vienna, VA 22182 (US).			Published <i>Without international search report and to be republished upon receipt of that report.</i>

(54) Title: A HIGH PERFORMANCE INTEROPERABLE NETWORK COMMUNICATIONS ARCHITECTURE (INCA)

(57) Abstract

An interoperable, software only network communications architecture (INCA) is presented that improves the internal throughput of network communicated data of workstation and PC class computers at the user level, application program level, by 260 % to 760 %. The architecture is unique because it is interoperable with all existing programs, computers and networks requiring minimal effort to set up and use. INCA operates by mapping network data between the application and operating address space without copying the data, integrating all protocol execution into a single processing loop in the application address space, performing protocol checksumming on a machine word size of data within the protocol execution loop, and providing an application program interface very similar to existing application program interfaces. The network interface driver functions are altered to set up network data transfers to and from the application address space without copying of the data to the OS address space, while buffer management, application to message multiplexing/demultiplexing and security functions are also being performed by the modified network interface driver software. Protocols are executed in the application address space in a single integrated protocol processing loop that interfaces directly to the INCA NI driver on one end and to the application on the other end in order to minimize the amount of times that network communicated data must travel across the internal memory bus. A familiar looking application program interface is provided that differs only slightly from existing application program interfaces which allows existing applications to use the new software with a minimum of effort and cost.



BEST AVAILABLE COPY

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece	ML	Mali	TR	Turkey
BG	Bulgaria	HU	Hungary	MN	Mongolia	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MR	Mauritania	UA	Ukraine
BR	Brazil	IL	Israel	MW	Malawi	UG	Uganda
BY	Belarus	IS	Iceland	MX	Mexico	US	United States of America
CA	Canada	IT	Italy	NE	Niger	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NL	Netherlands	VN	Viet Nam
CG	Congo	KE	Kenya	NO	Norway	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NZ	New Zealand	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	PL	Poland		
CM	Cameroon	KR	Republic of Korea	PT	Portugal		
CN	China	KZ	Kazakhstan	RO	Romania		
CU	Cuba	LC	Saint Lucia	RU	Russian Federation		
CZ	Czech Republic	LI	Liechtenstein	SD	Sudan		
DE	Germany	LK	Sri Lanka	SE	Sweden		
DK	Denmark	LR	Liberia	SG	Singapore		
EE	Estonia						

BEST AVAILABLE COPY

**A HIGH PERFORMANCE INTEROPERABLE
NETWORK COMMUNICATIONS ARCHITECTURE
(INCA)**

FIELD OF THE INVENTION

This invention relates generally to computer network communications. More particularly, the present invention relates to a method to improve the internal computer throughput rate of network communicated data.

BACKGROUND OF THE INVENTION

Network technology has advanced in the last few years from transmitting data at 10 million bit per second (Mbps) to near 1 Gigabit per second (Gbps). At the same time, Central Processing Unit (CPU) technology inside computers has advanced from a clock rate of 10 Million cycles (Hertz) per second (MHz) to 500 MHz. Despite the 500% to 1000% increase in network and CPU capabilities, the execution rate of programs that receive data over a network has only increased by a mere 100%, to a rate of approximately 2 Mbps. In addition, the internal computer delays associated with processing network communicated data have decreased only marginally despite orders of magnitude increase in network and CPU capabilities. Somewhere between the network interface (NI) and the CPU, the internal hardware and software architecture of computers is severely restricting data rates at the application program level and thereby negating network and CPU technology advances for network communication. As a result, very few network communication benefits have resulted from the faster network and CPU technologies.

Present research and prototype systems aimed at increasing internal computer throughput and reducing internal processing delay of network communicated data have all done so without increasing application level data throughput, or at the expense of interoperability, or both. For

1 purposes of this specification, network communicated data includes all matter that is
2 communicated over a network. Present research solutions and custom system implementations
3 increase data throughput between the NI of the computer and the network. However, the data
4 throughput of the application programs is either not increased, or is only increased by requiring
5 new or highly modified versions of several or all of the following: application programs,
6 computer operating systems (OSs), internal machine architectures, communications protocols
7 and NIs. In short, interoperability with all existing computer systems, programs and networks
8 is lost.

9 A range of problems associated with network operations is still present. The present state
10 of the art of increasing the performance of internal computer architectures:

- 11 1. Prevents computers from utilizing the tremendous advances in network and CPU
12 technologies;
- 13 2. Fails to solve the problem by focusing mainly on NI to network transfer rate
14 increases rather than on increasing network communicated data throughput at the
15 application program level;
- 16 3. Severely restricts computer network communicated data throughput at the
17 application program level to a fraction of existing low speed network and CPU
18 capabilities;
- 19 4. Prevents the use of available and the implementation of new computer applications
20 that require high network communicated data throughput at the application program
21 interface or low internal computer processing delays;
- 22 5. Requires a massive reinvestment by the computer user community in new
23 machines, software programs and network technologies because of a lack of
24 interoperability with existing computer systems and components.

25 A need therefore exists for higher network communicated data throughput inside

1 computers to allow high data rate and low processing delay network communication while
2 maintaining interoperability with existing application programs, computers, OSs, NIs, networks
3 and communication protocols.

4 SUMMARY OF THE INVENTION

5 It is an object of the present invention to provide high network communicated data
6 throughput and low network communicated data processing delay inside computers.

7 It is a further object of the present invention to maintain interoperability with all existing
8 computer programs, computers, OSs, NIs, networks and communication protocols.

9 It is a further object of the present invention to be useable on all existing personal
10 computer (PC) and workstation (WS) class computers, as well as on most other, greater
11 capability computer systems with only minor software modifications to the existing application
12 programs and/or OS.

13 It is a further object of the present invention to dramatically increase the network
14 communicated data throughput at the application level (not just NI to network).

15 It is a further object of the present invention to dramatically increase the network
16 communicated data throughput at the application level (not just NI to network level) for small
17 messages (less than or equal 200 bytes) and for large messages.

18 It is a further object of the present invention to speed up communication protocol
19 processing for all levels and types of protocols.

20 It is a further object of the present invention to reduce the amount of times network
21 communicated data is sent across the internal computer memory bus.

22 It is a further object of the present invention to increase the performance of non
23 networking applications on networked computers by processing network management messages
24 and messages not addressed to the machine at least four times faster than presently processed.

25 It is a further object of the present invention to be interoperable with existing computer

1 systems and network components and to not require costly changes to these components to
2 improve performance.

3 It is a further object of the present invention to require only minor software changes to
4 application program interfaces (APIs) or NI drivers.

5 It is a further object of the present invention to be installed, tested and operational in a
6 short amount of time, i.e., minutes to hours.

7 It is a further object of the present invention to provide for application (not just NI to and
8 from the network) throughput of network communicated data at high speed network transmission
9 rates.

10 It is a further object of the present invention to enable the use and implementation of
11 application programs that require high application level network communicated data throughput
12 and low internal computer network communicated data handling delay.

13 It is a further object of the present invention to allow the utilization of high speed network
14 and CPU technologies by enabling applications to process data at high speed network rates.

15 The present invention is a library of programs comprising three main programs integrated
16 into one software library: a computer NI driver, an integrated protocol processing (IPP) loop and
17 an API. The INCA NI driver comprises software that controls the NI hardware and transfers the
18 network messages and data in the messages from the network to the computer's memory. The
19 IPP loop software performs communication protocol processing functions such as error handling,
20 addressing, reassembly and data extraction from the network message. The API software passes
21 the network communicated data sent via the network to the application program that needs the
22 network communicated data. The same process works in reverse for transmitting network
23 communicated data from the application over the network to a remote computer. The existing
24 computer components, e.g., the NI, CPU, main memory, direct memory access (DMA) and OS,
25 are used with control or execution of these resources being altered by the INCA software

1 functions.

2 The advantage of using the present invention is that the existing, inefficient network
3 communicated data handling is greatly enhanced without requiring any additional hardware or
4 major software modifications to the computer system. INCA speeds up internal network
5 communicated data handling to such a degree that data can be transferred to and from the
6 application programs, via the network interface, at speeds approaching that of high speed
7 network communicated data transmission rates. The CPU is required to operate more frequently
8 on more data, utilizing the increased CPU capabilities. The present invention greatly reduces the
9 number of times network communicated data must be transferred across the internal computer
10 memory bus and greatly speeds up the processing of communication protocols, with particular
11 improvement in the checksumming function.

12 BRIEF DESCRIPTION OF THE DRAWINGS

13 **Figure 1** shows an overview of a typical existing network communication system.

14 **Figure 2** shows an overview of the INCA network communication system.

15 **Figure 3** shows an overview of the endpoint mechanism.

16 **Figure 4a** shows examples of the typical non INCA, non IPP for-loops used for protocol
17 processing.

18 **Figure 4b** shows an example of a single, integrated INCA IPP for-loop used for protocol
19 processing.

20 **Figure 5** shows the INCA IPP stages of protocol execution.

21 **Figure 6** shows the INCA IPP method of integrating various protocols into a single execution
22 loop.

23 **Figure 7** shows the alternative "system calls" comprising INCA's API.

24 **Figure 8** shows INCA's performance improvement on WS class computers.

25 **Figure 9** shows INCA's small message size performance improvement on PC class computers.

1 **Figure 10** shows INCA's performance improvement with all standard message sizes on PC class
2 computers.

3 **Figure 11** shows INCA's management and control flow.

4 **DETAILED DESCRIPTION OF THE INVENTION**

5 Referring to **Figure 1**, an overview of a typical existing network communication system
6 inside a computer is shown. When a network message **102** is received by the network interface
7 (NI) **104**, the NI **104** sends an interrupt signal to the operating system (OS) **106**. The network
8 communicated data is then copied from the NI **104** into the OS message buffers **110** which are
9 located in the OS memory address space **108**. Once the network communicated data is available
10 in the OS message buffers **110**, the OS **106** typically copies the network communicated data into
11 the Internet Protocol (IP) address space **112**. Once IP processing is completed, the network
12 communicated data is copied to the User Datagram Protocol (UDP)/Transport Control Protocol
13 (TCP) address space **114**. Once the UDP processing is completed, if any additional protocols
14 are used external to the application program, the network communicated data is copied to the
15 Other Protocol address space **116** where the network communicated data is processed further.
16 The network communicated data is then copied to the application program interface (API)
17 address space **118**. Finally, the application reads the network communicated data which requires
18 another copy of the network communicated data into the application program memory address
19 space **120**. As illustrated in **Figure 1**, copying of the network communicated data occurs
20 numerous times in the normal course of present operations.

21 Referring to **Figure 2**, an overview of the INCA network communication system is
22 shown. Contrasting INCA to the typical network communication system in **Figure 1**, it is
23 evident INCA eliminates several data copying steps and as a result, INCA performs in a more
24 efficient manner for increasing a network's data throughput rate. In addition, INCA implements
25 several other efficiency improving steps which will be discussed later.

1 As shown in **Figure 2**, the present invention comprises three main software components:
2 an INCA NI driver **202**, an INCA IPP (execution loop) **204** and an INCA API **206**. These
3 components reside inside computer message buffers **208** together with the current computer
4 software components such as application programs **210**, operating systems (OS) **212**,
5 communication protocols, and the current computer hardware components such as the NI **214**,
6 system memory bus **216** and **218**, OS memory address space **220** and application program
7 memory address space **224** and one or more CPUs, disks, etc.

8 The first component, the INCA NI driver **202**, is a software set of programming language
9 functions that supports direct access to the NI **214** and performs the following functions:

- 10 1. Controls the NI device **214** and other involved devices (e.g., DMA) to set up a
11 transfer of network messages **222** from the NI **214** to OS memory address space **220**;
- 12 2. Manages the NI **214** to computer memory transfer;
- 13 3. Demultiplexes incoming network messages to the proper recipient (i.e.,
14 application);
- 15 4. Provides protection of network communicated data from different applications;
- 16 5. Transfers the network communicated data to the application program memory
17 address space **224**;
- 18 6. Interfaces to the INCA IPP **204** to control protocol execution;
- 19 7. Interfaces to the INCA API **206** to control application program network access;
- 20 8. Relinquishes any control over the NI **214** upon completion of message handling.

21 These eight functions are performed by the INCA NI driver **202** component for reception of
22 network messages. In the case of the transmission of network communicated data from one
23 computer to another computer over a network, the eight functions are performed in reverse order.
24 Since the transmitting case is the same as the receiving case, it is not discussed separately. The
25 following description of the INCA NI driver functions is for the receiving case. The INCA NI

1 driver component may include software linked to the INCA software library which is not
2 typically considered NI device driver code.

3 The first two functions, control and management of transferring network communicated
4 data from the NI device to internal computer memory (i.e., random access memory - RAM), or
5 some other type of memory (e.g. cache, hard disk) are initiated when a message arrives at the
6 computer from a network connection. The NI hardware signals the arrival of a message,
7 typically a hardware interrupt. The message arrival notification signal is received by the INCA
8 NI driver 202. Upon receipt of a message arrival notification, the INCA NI driver 202 takes
9 over control of the NI device (e.g., Asynchronous Transfer Mode (ATM) network card), and sets
10 up the registers, firmware, etc., of the device to transfer the message from the device to main
11 memory. Transferring the message or network communicated data is in response to the call
12 functions of either an application program interface, an application program, a network interface
13 device or a network interface driver.

14 The transfer can be accomplished via two main methods, via DMA or programmed
15 input/output (PIO). In the case of DMA transfers of network communicated data between the
16 NI 214 and OS memory address space 220, the INCA NI driver 202 sets up memory and NI
17 device addresses, message buffer sizes/alignments/addresses, and signals the start and completion
18 of every transfer. If an error occurs, the INCA NI driver 202 attempts to resolve the error
19 through such actions as reinitializing a DMA transfer or repeating transfers. At the completion
20 of DMA transfers, the INCA NI driver 202 releases control of any DMA and NI 214, releases
21 any allocated message buffer memory 208, and ceases execution.

22 In the case of PIO, the CPU transfers every byte of network communicated data from the
23 NI to computer memory. The INCA NI driver 202 provides the necessary parameters, memory
24 and NI addresses, transfer sizes and buffer sizes/alignments/addresses for the CPU to transfer the
25 network communicated data to computer memory.

1 In the preferred embodiment, the OS 212 manages the address mapping between the
2 virtual addresses of message buffers specified by an application and the physical addresses
3 required for actual transmission and reception. In an alternative embodiment, the application
4 program manages the address mapping. In yet another embodiment, hardware, such as the NI
5 214, manages the address mapping.

6 The OS 212 performs virtual memory (VM) management through the use of a memory
7 mapping function such as the UNIX OS mmap() function which maps the message buffers 208
8 in OS memory address space 220 to the application program memory address space 224. Virtual
9 to physical address translations are therefore handled in the existing OS manner. To enable the
10 OS 212 to perform VM management and address translation, the INCA NI driver 202 must
11 allocate message buffers 208 in the OS memory address space 220 initially, as well as in the
12 application program memory address space 224 to allow the OS 212 to make the required
13 mappings and perform its VM management functions. The INCA NI driver 202 performs these
14 functions as soon as the message arrival signal is received by the INCA NI driver 202. The
15 address locations of the message buffers 208 containing the network communicated data are
16 therefore mapped to the VM locations in the IPP address space 218, with only one physical
17 memory location, hence no copying of the network communicated data is required.

18 Message buffer management and DMA management are performed by the INCA NI
19 driver 202. The INCA NI driver 202 allocates buffer space when an application 210 calls the
20 INCA NI driver 202 with an INCA open() call which opens the INCA NI driver 202 to initialize
21 the DMA transfer. The INCA NI driver 202 receives the NI message interrupt signal and starts
22 the INCA NI driver 202 which causes message buffer allocation to occur in message buffers 208
23 and in IPP address space 226. The INCA NI driver 202 uses the 4 KB memory page size
24 provided by most OS VM systems, and allocates message buffers in 2 KB increments. Each
25 message is aligned to the beginning of the 2 KB buffer with a single message per buffer for

1 messages smaller than 2 KB. For messages larger than 2 KB, multiple buffers are used
2 beginning with the first and intermediate buffers filled from start to end. The last buffer contains
3 the remnant of the message starting from the 2 KB buffer VM address position. For messages
4 equal to or less than 2 KB, one buffer is allocated and the contents are aligned with the first byte
5 placed at the start of the 2 KB buffer address space.

6 In order to make the mapping from the OS space to user space easier and in order to avoid
7 implementing more memory management functionality into INCA, the message buffers are
8 "pinned" or assigned to fixed physical memory locations in either the application or OS address
9 space. The application specifies message buffers using offsets in the buffer region, which the
10 INCA NI driver 202 can easily bounds-check and translate. By using fixed physical memory
11 locations, the INCA NI driver 202 will not issue illegal DMA access. Since INCA has complete
12 control over the size, location, and alignment of physical buffers, a variety of buffer management
13 schemes are possible.

14 All buffers may be part of a system-wide pool, allocated autonomously by each domain
15 (e.g., applications, OS), located in a shared VM region, or they may reside outside of main
16 memory on a NI device. Physical buffers are of a fixed size to simplify and speed allocation.
17 The INCA NI driver memory management is immutable, it allows the transparent use of page
18 remapping, shared virtual memory, and other VM techniques for the cross-domain transfer of
19 network communicated data. Virtual copying with the mmap() function is used to make domain
20 crossings as efficient as possible, by avoiding physical memory bus transfer copying between
21 the OS 212 and application program memory address space 224.

22 The third function of the INCA NI driver 202 is message demultiplexing (for receiving)
23 and multiplexing (for transmitting). Not all applications on a machine may be using the INCA
24 software to communicate over the network. There may be a mix of INCA and non INCA
25 communicating applications in which case the INCA NI driver 202 must also route messages to

1 the non INCA NI driver or the non INCA protocol processing software, or to some other non
2 INCA software. The INCA NI driver 202 maintains a list of INCA application program
3 addresses known as endpoints. Endpoints provide some of the information required to carry out
4 the INCA NI driver component functions.

5 Referring to **Figure 3**, an overview of the endpoint mechanism is shown. Endpoints 302
6 bear some resemblance to conventional sockets or ports. A separate endpoint is established and
7 maintained for each application and each network connection for each application. For
8 applications without INCA endpoint addresses, non INCA networking applications, the INCA
9 NI driver passes the message arrival notification to the non INCA NI driver.

10 Each application that wishes to access the network first requests one or more endpoints
11 302 through the INCA alternative API "system calls". The INCA NI driver then associates a set
12 of send 304, receive 306, and free 308 message queues with each endpoint through the use of two
13 INCA "system calls", `inca_create_endpoint()` and `inca_create_chan()`. The application program
14 memory address space 300 contains the network communicated data and the endpoint message
15 queues (endpoint send/receive free queues 304, 306, 308) which contain descriptors for network
16 messages that are to be sent or that have been received.

17 In order to send, an application program composes a network message in one or more
18 transmit buffers in its address space and pushes a descriptor onto the send queue 304 using the
19 INCA API "system calls". The descriptor contains pointers to the transmit buffers, their lengths
20 and a destination tag. The INCA NI driver picks up the descriptor, allocates virtual addresses
21 for message buffers in OS memory address space and sets up DMA addresses. The INCA NI
22 driver then transfers the network communicated data directly from the application program
23 memory address space message buffers to the network. If the network is backed up, the INCA
24 NI driver will simply leave the descriptor in the queue and eventually notifies the user
25 application process to slow down or cease transmitting when the queue is near full. The INCA

1 NI driver provides a mechanism to indicate whether a message in the queue has been injected
2 into the network, typically by setting a flag in the descriptor. This indicates that the associated
3 send buffer can be reused.

4 When the INCA NI driver receives network communicated data, it examines the message
5 header and matches it with the message tags to determine the correct destination endpoint. The
6 INCA NI driver then pops free buffer descriptors off the appropriate free queue 308, translates
7 the virtual addresses, transfers the network communicated data into the message buffers in OS
8 memory address space, maps the memory locations to the application program memory address
9 space and transfers a descriptor onto the receive queue 306. Each endpoint contains all states
10 associated with an application's network "port".

11 Preparing an endpoint for use requires initializing handler-table entries, setting an
12 endpoint tag, establishing translation table mappings to destination endpoints, and setting the
13 virtual-memory segment base address and length. The user application program uses the API
14 routine calls "ioctl()" and "mmap()" to pass on any required endpoint data and provide the VM
15 address mapping of the OS message buffers to the application program memory address space
16 locations. Once this has been achieved, the user application is prepared to transmit and receive
17 network communicated data directly into application program memory address space. Each
18 endpoint 302 is associated with a buffer area that is pinned to contiguous physical memory and
19 holds all buffers used with that endpoint. Message descriptors contain offsets in the buffer area
20 (instead of full virtual addresses) which are bounds-checked and added to the physical base
21 address of the buffer area by the INCA NI driver. In summary, endpoints and their associated
22 INCA NI driver "system calls" set up an OS-Bypass channel for routing network communicated
23 data address locations to and from memory to the correct applications.

24 Providing some security is the fourth function performed by the INCA NI driver. To
25 assure that only the correct applications access the message data, application program identifiers

1 to endpoints and endpoints to message data mappings are maintained. An application can only
2 access message data in the endpoint message queues where the identifiers of endpoint(s) of
3 message queues matches the identifiers of endpoints for that application. Any access to network
4 communicated data must come from the intended recipient application or in the case of
5 transmitting network communicated data, access to the network communicated data must come
6 from the transmitting application.

7 Once the network communicated data transfer is set up and demultiplexing of messages
8 is complete, the INCA NI driver performs the function of transferring the network communicated
9 data from the OS memory address space to the receiving application program memory address
10 space. This transfer is required since all present NI devices come under the ownership of the
11 computer OS and any network communicated data transferred via a NI device is allocated to the
12 OS virtual or physical memory address space. INCA makes this transfer without requiring any
13 movement or copying of the network communicated data, thereby avoiding costly data copying.
14 The transfer is made via a mapping of the memory addresses of the network communicated data
15 within the OS memory address space to memory (addresses) within the application program
16 memory address space.

17 For UNIX OS based systems, the UNIX mmap() function is used by the INCA NI driver
18 to perform the transferring of network communicated data to the application address space,
19 mapping the addresses of the network data in the OS address space to the application address
20 space.

21 The sixth function of the INCA NI driver is to interface to INCA's second component,
22 the IPP loop software. Once network communicated data is available in the computer's memory,
23 the INCA NI driver notifies the IPP software that network communicated data is available for
24 protocol processing. The notification includes passing a number of parameters to provide needed
25 details for the IPP software. The parameters include the addresses of the network communicated

1 data and the endpoints to determine the recipient application program.

2 The IPP component of the invention is an extension of Integrated Layer Processing (ILP),
3 performing the functions of communications protocol processing. IPP includes protocols above
4 the transport layer, including presentation layer and application layer protocol processing and
5 places the ILP loop into one integrated execution path with the INCA NI driver and API
6 software. In current systems, communication protocol processing is conducted as a part of and
7 under the control of the OS in OS memory address space. Each protocol is a separate process
8 requiring all the overhead of non integrated, individual processes executed in a serial fashion.
9 Existing and research implementations do not integrate ILP with an NI OS-Bypass message
10 handler and driver, do not integrate protocol processing into a single IPP loop, nor do they
11 execute protocols in user application program memory space. Protocol execution by the existing
12 OSs and under the control of the OS are not used by INCA's IPP component. INCA's IPP
13 performs protocol execution using the INCA software library implementations of the protocols
14 linked to the application in the application program memory address space.

15 Referring to **Figure 4a**, a depiction of a "C" code example of typical protocol processing
16 code is shown. Before the code can be executed, the network communicated data must be copied
17 to each protocol's memory address space. When the code is compiled to run on a reduced
18 instruction set computer (RISC) CPU, the network message data manipulation steps results in
19 the machine instructions noted in the comments. First, the protocol software process, e.g., the
20 Internet Protocol (IP) software, is initialized and the network communicated data is copied from
21 the OS message buffer memory area to the IP process execution memory area. Each time a word
22 of network communicated data is manipulated, the word is loaded and stored into memory.
23 Upon completion of the first protocol, the second protocol process, e.g., the TCP software, is
24 initialized and the network communicated data is copied to this protocol's execution area in
25 memory. Once again, each time a word of network communicated data is manipulated, the word

1 is loaded and stored. This process continues until all protocol processing is complete.

2 Referring to **Figure 4b**, the INCA system with the IPP method is shown, where each
3 word is loaded and stored only once, even though it is manipulated twice. Each protocol's
4 software execution loop is executed in one larger loop, eliminating one load and one store per
5 word of data. This is possible because the data word remains in a register between the two data
6 manipulations. Integrating the protocol processing for-loops results in the elimination of one
7 load and one store per word of network communicated data. The IPP method of performing all
8 protocol processing as one integrated process, also eliminates the copying of all network
9 communicated data between the initialization and completion of each separate communications
10 protocol used (e.g., copying the data to IP protocol address space, then copying the data to UDP
11 or TCP protocol address space, etc.).

12 In addition, the INCA IPP protocol processing uses an optimized protocol checksum
13 processing routine that calculates checksums on a word (e.g., 4 to 8 bytes depending upon the
14 machine) of network communicated data at a time, rather than the existing method of one byte
15 at a time. INCA's IPP checksum calculation is roughly four times faster than existing checksum
16 calculations. For small message sizes of less than or equal to 200 bytes, which comprise some
17 90% or more of all network messages, INCA's IPP checksum routine greatly speeds up the
18 processing of small messages since checksum calculation is the majority of calculations required
19 for small messages.

20 The IPP component divides protocol processing of network messages into three
21 categories: data manipulation - reading and writing application data, header processing - reading,
22 writing headers and manipulating the headers of protocols that come after this protocol, and
23 external behavior - passing messages to adjacent layers, initiating messages such as
24 acknowledgments, invoking non-message operations on other layers such as passing congestion
25 control information, and updating protocol state such as updating the sequence number

1 associated with a connection to reflect that a message with the previous number has been
2 received.

3 Referring to **Figure 5**, the INCA IPP component executes the protocols in three stages
4 in a processing loop: an initial stage **502**, a data manipulation stage **504** and a final stage **506**.
5 The initial stages of a series of layers are executed serially, then the integrated data manipulations
6 take place in one shared stage and then the final stages are executed serially. Interoperability
7 with existing protocol combinations such as IP, TCP, UDP and External Data Representation
8 (XDR) combinations requires the IPP software to contain some serial protocol function
9 processing of the network communicated data in order to meet the data processing ordering
10 requirements of these existing protocols. Message processing tasks are executed in the
11 appropriate stages to satisfy the ordering constraints. Header processing is assigned to the initial
12 stage. Data manipulation is assigned to the integrated stage. Header processing for sending and
13 external behavior (e.g., error handling) are assigned to the final stage.

14 Referring to **Figures 5 and 6**, INCA's IPP method of integrating multiple protocols is
15 shown. The protocols of protocol A **610** and protocol B **620** are combined and INCA's IPP
16 method integrates the combination of protocol A and B **628**. The initial stages **612, 622** are
17 executed serially **502** (as shown in Figure 5), then the integrated data manipulation **504** is
18 executed **614, 624**, and then the final stages **616, 626** are executed serially **506**. Executing the
19 tasks in the appropriate stages ensures that the external constraints protocols impose on each
20 other cannot conflict with their internal constraints.

21 The ILP software starts up directly after reception of network communicated data into
22 the message buffer. It does the integrated checksumming on the network communicated data in
23 the initial stage **502**, protocol data manipulations and Host/Network byte order conversions in
24 the middle integrated stage **504**, and TCP type flow control and error handling in the final stage
25 **506**. The concept of delayed checksumming has been included in the loop. In the case of IP

1 fragments, the checksumming is done only after reassembly. Message fragments are transmitted
2 in the reverse order, i.e., the last fragment is transmitted first, to make the time of checksumming
3 less in the case of UDP. Once the data processing is complete, the packets are multiplexed to
4 the corresponding protocol ports set up by the API.

5 The IPP protocol library software consists of software functions that implement the
6 protocol processing loop and other pieces of protocol control settings such as fragmentation, and
7 in the case of TCP, maintaining the window, setting up time-out and retransmission etc. The
8 TCP library has been implemented with a timer mechanism based on the real-time clock and a
9 Finite State Machine (FSM) implementation.

10 The INCA IPP component integrates protocol processing into one process which executes
11 in the application program's memory address space. Consequently, the number of copies of the
12 network communicated data to and from memory are greatly reduced, as can be seen by
13 comparing Figures 1 and 2. Thus the speed and efficiency with which data can be accessed and
14 used by an application is increased. These repeated transfers across the CPU/memory data path
15 frequently dominate the time required to process a message, and thus the network
16 communications throughput. The IPP component therefore speeds up network communicated
17 data processing through the elimination of time consuming memory bus transfers of the network
18 communicated data. With the use of the IPP loop's high performance protocol checksum
19 software, protocol processing time of network communicated data is greatly reduced, providing
20 a large part of the performance improvement of the invention.

21 Interoperability requires the IPP loops to use and execute existing protocols as well as
22 future protocols. The INCA software libraries accommodate the integration of all existing
23 protocols and future protocols into the IPP component of INCA and provide integration with the
24 INCA NI driver component functions.

25 The seventh function of the INCA NI driver is to interface to INCA's third component,

1 the API. This interface provides the application with network access for sending data and also
2 sets up application address space parameters for transfer of network data to an application.

3 The API component of the invention provides the interface between the existing
4 application programs and the new INCA components. The API allows existing applications to
5 call on the INCA software to perform network communicated data handling in the new, high
6 performance manner. The API limits the changes required to existing application programs to
7 minor name changes to their current API and thereby provides interoperability with existing
8 application programs. The INCA API allows the application to: open a network connection by
9 opening the NI device, specify parameters to the INCA NI driver, specify the protocols to use
10 and their options, set the characteristics of the data transfer between the application and the
11 network using IPP and the INCA NI driver, and detect the arrival of messages by polling the
12 receive queue, by blocking until a message arrives, or by receiving an asynchronous notification
13 on message arrival (e.g., a signal from the INCA NI driver). The API also provides low level
14 communication primitives in which network message reception and transmission can be tested
15 and measured.

16 Referring to **Figure 7**, the INCA API uses alternative "system call" type programming
17 code structures **701** to **712** in place of the current OS system calls such as socket(), connect(),
18 listen() and bin(). The alternative calls are used to bypass the current OS system calls. In an
19 alternative embodiment, the operating system can include the alternative system calls. The new
20 system calls initiate the INCA IPP and INCA NI driver software library programs to provide the
21 necessary API functionality. The API set of system calls **701** to **712** simplifies the application
22 programming required to use the invention, the INCA software library, to renaming the existing
23 calls by placing "inca_" in front of the existing calls. As depicted in **Figure 7**, the API provides
24 the basic commands like "open()", "close()", "read()", "write()", etc., similar to existing system
25 networking APIs.

1 The "open()" call 701, 702 and 709 will perform the following for the user:

- 2 1. Open the device for operation;
- 3 2. Create the OS Bypass structure and set up a DMA channel for user to network
- 4 transfer;
- 5 3. Map the communication segment from the driver buffer to the user buffer;
- 6 4. Open an unique channel for communication between two communicating entities;
- 7 5. Fill up this buffer (incabuffer), which will be used in calls to read, write, etc.

8 The "close()" call 703, 704 and 710 will perform the following for the user:

- 9 1. Free storage allocated for the buffers;
- 10 2. Destroy the communication channel;
- 11 3. Unmap the communication segment;
- 12 4. Remove the DMA mapping;
- 13 5. Close the device used by that particular application.

14 The "read()" call 705 and 706, with a pointer to "incabuffer" as the parameter, will perform the
15 following for the user:

- 16 1. Receive any pending packet from the INCA device, which has been transferred via
- 17 DMA;
- 18 2. Pass the received packet (if not fragmented) through the IPP loop; or if
- 19 fragmented, pass the received packet through a special IPP loop (inlined) which will
- 20 do IPP, but will keep track of the fragments and pass it on only when the packet is
- 21 complete and reassembled;
- 22 3. Close the read call (the packet is ready for delivery to the application).

23 The "write()" call 707 and 708, with a pointer to "incabuffer" as the parameter, will perform the
24 following for the user:

- 25 1. Create the header for IP/UDP/TCP;

- 1 2. Perform IPP processing on the packet;
- 2 3. Fragment the packet if it is too large for UDP or IP by passing the received packet
- 3 through a special IPP loop (inlined), which will keep track of the fragments and pass
- 4 it on only when the packet fragmentation is complete;
- 5 4. Pass on the IP packets for transmission onto the NI.

6 Parameters passed by the application to the IPP and NI driver components of INCA inside the
7 () of the calls include application message identifiers and endpoints. This parameter allows the
8 IPP and INCA NI driver components to multiplex and demultiplex messages to the intended
9 applications or network address. A more enhanced API could include calls or parameters within
10 calls to set up a connection using TCP and also functions that help in the implementation of
11 client/server applications like "listen" etc.

12 Although INCA's API can be located anywhere between the networking application
13 program and any protocol of the protocol stack, the API is typically located between the
14 application and the INCA IPP component. In current OSs, the API typically sits between the
15 session layer (e.g., socket system calls) and the application. Ideally, the API sits between the
16 application and all other communication protocols and functions. In current systems and
17 application programs, many times the application also contains the application layer protocol
18 (i.e., Hypertext Transport Protocol - HTTP), the presentation layer protocol functions (i.e., XDR
19 like data manipulation functions), and only the socket or streams system call is the API. This
20 is not necessarily ideal from a user perspective. By integrating presentation and application
21 protocol functions into the application, any change in these functions necessitates an application
22 program "upgrade" at additional procurement, installation time and maintenance cost. INCA can
23 incorporate all the application, presentation and session (replacement for socket and streams
24 calls) functions into the IPP loop. In the future, this can even be accomplished dynamically, at
25 run time, through application selection of the protocol stack configuration.

1 The API provides the application the link to utilize the INCA high performance network
2 communication subsystem as opposed to using the existing system API and existing network
3 communicated data processing subsystem. The existing system API could also be used to
4 interface to the INCA IPP and INCA NI driver components if the existing system API is
5 modified to interface to the INCA IPP protocol processing and INCA NI driver network interface
6 and instead of the existing system protocol processing and network interface software.

7 The final function is relinquishing control of the NI device. The INCA NI driver uses an
8 alternative "system call", `inca_closedev()`, in place of the current OS system call to close the NI
9 device and to relinquish control of the NI device. When the NI device has no more network
10 communicated data to be transferred, the INCA NI driver relinquishes control of the NI device
11 to the computer's OS so that other software can use the NI device. Hardware or software
12 interrupts are typically used to signal that the NI device has no more network communicated data
13 to transfer. Upon detection of the no more network communicated data to transfer signal, the
14 INCA NI driver sets the end memory address of the network communicated data buffers. For
15 the mapping of the network communicated data into the application address space, the INCA NI
16 driver performs any required message buffer alignment and passes the network communicated
17 data address range to the IPP software. The NI device is set to a known state and the OS is
18 notified that the device is available to be scheduled for use by other software processes.

19 To illustrate the workings and ease of use of the invention, the following description is
20 provided. The INCA software library is loaded unto the computer's hard disk. If the machine's
21 NI device drivers are implemented as loadable modules, no NI device driver modifications are
22 necessary. If the INCA NI driver is integrated into the OS without being a separate module, the
23 INCA NI driver software is integrated into the OS through a recompilation of the OS. This does
24 not alter the operation of existing programs or the OS, but only adds the INCA NI interface. For
25 those networking applications that will use the INCA software library for network message

1 handling, the API system calls are changed to the INCA API system calls. This procedure can
2 be accomplished via a number of approaches. Once accomplished, all is complete and system
3 operation can resume. These two steps, INCA NI driver insertion and renaming the API calls,
4 provide a system by system approach to using the INCA networking software. System vendors
5 or system administrators are the most likely candidates to use this method. An alternative
6 approach to the above steps is to integrate the entire library into the applications desiring to
7 perform networking with the INCA software. This provides an application by application
8 approach to using INCA. Application program vendors or individual users are the most likely
9 candidates to use this method. Either way, the entire procedure can be accomplished in minutes
10 to hours depending upon the implementor's familiarity with the applications, OS and NI device
11 drivers. For existing applications that do not have their system calls modified, INCA allows the
12 traditional network system interfaces (e.g., sockets) with the applications. Referring to **Figure**
13 **11**, the order of events for receiving network communicated data over the network is shown.
14 Once system operation begins and network messages are received, the order of events for
15 receiving data over the network are as follows: the NI device driver receives a message arrival
16 notification from the NI hardware **1100**, typically via a hardware interrupt. The message arrival
17 notification signal is received by the OS and initiates the opening of the INCA enhanced NI
18 driver - the INCA NI driver **1102**. The INCA NI driver determines if the network message is for
19 an application that can use the INCA software library to receive messages **1104**. If the
20 application is not "INCA aware," control is handed over to the OS for further handling **1108**. If
21 the application can use INCA to communicate, the INCA NI driver takes control of the NI device
22 **1106** (e.g., Asynchronous Transfer Mode - ATM network card) and sets up the registers,
23 firmware, etc., of the device to transfer the network communicated data from the NI device to
24 internal computer memory **1110**. The INCA NI driver uses an alternative "system call" type
25 programming code structure, `inca_opendev()`, in place of the current OS system calls to take over

1 the device and set up the data transfer from the NI device to computer memory. The INCA
2 driver then uses the endpoint identifiers to demultiplex incoming messages to the recipient
3 application program 1112. The network message buffers in OS address space are mapped to the
4 recipient application's address space 1114. The INCA IPP software is configured and started for
5 protocol processing 1116. The IPP software performs protocol processing to extract the data
6 from the network message(s) 1118. Once the first IPP loop is completed, the application is
7 notified via the INCA API calls that data is ready for consumption 1120. The application then
8 processes the data 1122. If there are more messages to process, the IPP loop continues
9 processing and the application continues consuming the data 1124. When all messages have
10 been received 1126, the NI driver closes the NI device and relinquishes control of the device to
11 the OS 1128.

12 For transmission of data, the entire process occurs in reverse order and the application
13 uses the API calls to communicate with the IPP software to determine which protocols and
14 protocol options to use, sets up an endpoint by opening the INCA NI driver with the open() API
15 call, establishes an endpoint, sets the endpoint and driver DMA characteristics with INCA API
16 system calls such as ioctl(), and upon transmission completion, uses close() to close the INCA
17 NI driver. The IPP component executes the selected protocols and places the resulting network
18 communicated data into the send queue message buffers. The INCA NI driver ceases control of
19 the NI device and DMA resources with its "system calls" to the OS, maps the send queue in
20 application address space to the OS message buffers in OS address space using the function
21 mmap(), sets up and controls the DMA transfer from the OS message buffers to the NI device,
22 and upon completion, relinquishes control of the NI device and DMA resources.

23 The API calls are the method of communication between the three INCA components and
24 the existing application programs, OS and computer resources such as the NI and DMA devices.

25 RESULTS

1 Tests were conducted on commercially available systems, configured with commercial-
2 off-the shelf (COTS) software, NIs and a FastEthernet network. The INCA testbed consisted of
3 two machines connected via a 100 Mbps FastEthernet. INCA allows applications to process data
4 at rates greater than 10 Mbps, thereby a normal 10 Mbps Ethernet would have caused the
5 network to limit INCA performance. A SUN Microsystems UltraSPARC1 WS with a 143 MHz
6 UltraSPARC1 CPU, 64 MB of RAM, running Solaris 2.5.1 (also known as SUN OS 5.5.1), with
7 a SUN SBus FastEthernet Adapter 2.0 NI was connected via a private (no other machines on the
8 network) 100 Mbps FastEthernet to a Gateway 2000 PC with a 167 MHz Pentium Pro CPU, 32
9 MB of RAM, running the Linux 2.0 OS with a 3Com FastEtherlink (Parallel Tasking PCI 10/100
10 Base-T) FastEthernet NI. Messages of varying lengths from 10 bytes to the maximum allowable
11 UDP size of 65K bytes were sent back and forth across the network between the machines using
12 an Internet World Wide Web (WWW) browser as the application program on both machines.
13 This architecture uses the actual application programs, machines, OSs, NIs, message types and
14 networks found in many computing environments. The results should therefore have wide
15 applicability.

16 SUN UltraSPARC1 Workstation with and without INCA

17 Referring to **Figure 8**, the graph illustrates the fact that on a high performance WS class
18 computer, INCA outperforms the current system at application program network message
19 throughput by 260% to 760% depending upon the message size. Since 99% of TCP and 89%
20 of UDP messages are below 200 bytes in size, the region of particular interest is between 20 and
21 200 byte size messages.

22 Gateway 2000 Pentium Pro PC with and without INCA

23 Referring to **Figures 9 and 10**, the graphs illustrate that on a PC class computer, INCA
24 outperforms the current system at application program network message throughput by 260%
25 to 590%. **Figure 9** shows INCA's 260% to 275% performance improvement for message sizes

1 of 10 to 200 bytes. **Figure 10** shows that as message sizes get larger and larger, up to the
2 existing protocol limit of 65K bytes, INCA's performance improvement becomes larger and
3 larger reaching the maximum of 590% at a message size of 65K bytes.

4 Although the method of the present invention has been described in detail for purpose of
5 illustration, it is understood that such detail is solely for that purpose, and variations can be made
6 therein by those skilled in the art without departing from the spirit and scope of the invention.

7 The method of the present invention is defined by the following claims:

1 We claim:

2 1. A method for improving the internal computer throughput rate of network communicated data
3 comprising transferring network communicated data from a network interface device to an
4 application address space with only one physical copying of the data.

5 2. The method for improving the internal computer throughput rate of network communicated
6 data of claim 1, wherein the copying of the data occurs in response to a call by an application
7 program interface which bypasses the operating system calls.

8 3. The method for improving the internal computer throughput rate of network communicated
9 data of claim 2, where the call functions of the application program interface are integrated into
10 the existing operating system.

11 4. The method for improving the internal computer throughput rate of network communicated
12 data of claims 1 or 2, further comprising the transfer of network communicated data from the
13 network interface device directly to the application address space.

14 5. The method for improving the internal computer throughput rate of network communicated
15 data of claim 4, further comprising the transfer of network communicated data from the network
16 interface device to application address space through an address mapping of the network
17 communicated data between the operating system address space and the application address
18 space.

19 6. The method for improving the internal computer throughput rate of network communicated
20 data of claim 5, wherein the transfer of network message data from the network interface device
21 to the application address space is controlled by the operating system.

22 7. The method for improving the internal computer throughput rate of network communicated
23 data of claim 5, wherein the transfer of network message data from the network interface device
24 to the application address space is controlled by the application program.

25 8. The method for improving the internal computer throughput rate of network communicated

1 data of claim 5, wherein the transfer of network message data from the network interface device
2 to the application address space is controlled by a hardware component.

3 9. The method for improving the internal computer throughput rate of network communicated
4 data of claim 5, wherein the transfer of network message data from the network interface device
5 to the application address space is controlled by the network interface device.

6 10. The method for improving the internal computer throughput rate of network communicated
7 data of claim 5, wherein the transfer of network message data from the network interface device
8 to the application address space is a direct memory access transfer.

9 11. The method for improving the internal computer throughput rate of network communicated
10 data of claim 10, further comprising reinitializing a direct memory access if an error occurs.

11 12. The method for improving the internal computer throughput rate of network communicated
12 data of claim 10, further comprising repeating a direct memory access transfer if an error occurs.

13 13. The method for improving the internal computer throughput rate of network communicated
14 data of claims 5, wherein the transfer of the network communicated data is a programmed
15 input/output transfer.

16 14. The method for improving the internal computer throughput rate of network communicated
17 data of claim 5, wherein the operating system of the computer manages the address mapping
18 between the virtual memory addresses and physical memory addresses of the network
19 communicated data in the operating system and application memory address spaces.

20 15. The method for improving the internal computer throughput rate of network communicated
21 data of claim 14, further comprising the network interface driver translating the address mapping
22 between the virtual memory addresses and physical memory addresses of the network
23 communicated data in the operating system and application memory address spaces.

24 16. The method for improving the internal computer throughput rate of network communicated
25 data of claim 14, further comprising the network interface driver demultiplexing network

1 messages and routing the network messages to the proper application.

2 17. The method for improving the internal computer throughput rate of network communicated
3 data of claim 16, further comprising the network interface driver examining the header of the
4 message to determine the correct destination point of the message.

5 18. The method for improving the internal computer throughput rate of network communicated
6 data of claim 16, further comprising the network interface driver maintaining a list of the
7 application endpoints.

8 19. The method for improving the internal computer throughput rate of network communicated
9 data of claim 14, further comprising the network interface driver providing security by permitting
10 only an intended recipient of the network communicated data to access the network
11 communicated data.

12 20. The method for improving the internal computer throughput rate of network communicated
13 data of claim 14, further comprising the network interface driver notifying and providing
14 parameters to an integrated protocol processing loop to allow an integrated protocol processing
15 loop to perform protocol processing on the network communicated data.

16 21. The method for improving the internal computer throughput rate of network communicated
17 data of claim 20, wherein the network interface driver sets end memory addresses of the message
18 buffers, aligns the message buffers and passes the range of the message buffers to the integrated
19 protocol processing loop.

20 22. A method for improving the internal computer throughput rate of network communicated
21 data comprising executing communication protocols in an integrated protocol processing loop.

22 23. The method for improving the internal computer throughput rate of network communicated
23 data of claim 22, further comprising linking the proper protocols to an application in the
24 application program memory address space.

25 24. The method for improving the internal computer throughput rate of network communicated

1 data of claim 22, further comprising the integrated protocol processing loop containing iterations
2 of serial and integrated data manipulations.

3 25. The method for improving the internal computer throughput rate of network communicated
4 data of claim 22, wherein header processing is performed during serial data manipulation, data
5 manipulation is performed during integrated data manipulation and header and external behavior
6 is performed during serial data manipulation.

7 26. A method for improving the internal computer throughput rate of network communicated
8 data comprising calculating communication protocol checksums one computer word at a time
9 within an integrated protocol processing loop.

10 27. The method for improving the internal computer throughput rate of network communicated
11 data of claim 26, wherein the size of a computer word is 32 bits.

12 28. The method for improving the internal computer throughput rate of network communicated
13 data of claim 26, wherein the size of a computer word is 64 bits.

14 29. A method for improving the internal computer throughput rate of network communicated
15 data comprising:

16 transferring network communicated data from a network interface device to an

17 application address space with only one physical copying of the data;

18 executing communication protocols in an integrated protocol processing loop;

19 calculating communication protocol checksums one computer word size of data at a time

20 within the integrated protocol processing loop; and

21 address mapping of the data occurs in response to call functions, where the operating

22 system's calls are bypassed.

23 30. The method for improving the internal computer throughput rate of network communicated
24 data of claim 29, wherein the call functions are call functions of an application program interface.

25 31. The method for improving the internal computer throughput rate of network communicated

1 data of claim 29, wherein the call functions are call functions of an application program.

2 32. The method for improving the internal computer throughput rate of network communicated

3 data of claim 29, wherein the call functions are call functions of a network interface device.

4 33. The method for improving the internal computer throughput rate of network communicated

5 data of claim 29, wherein the call functions are call functions of a network interface driver.

1 / 11

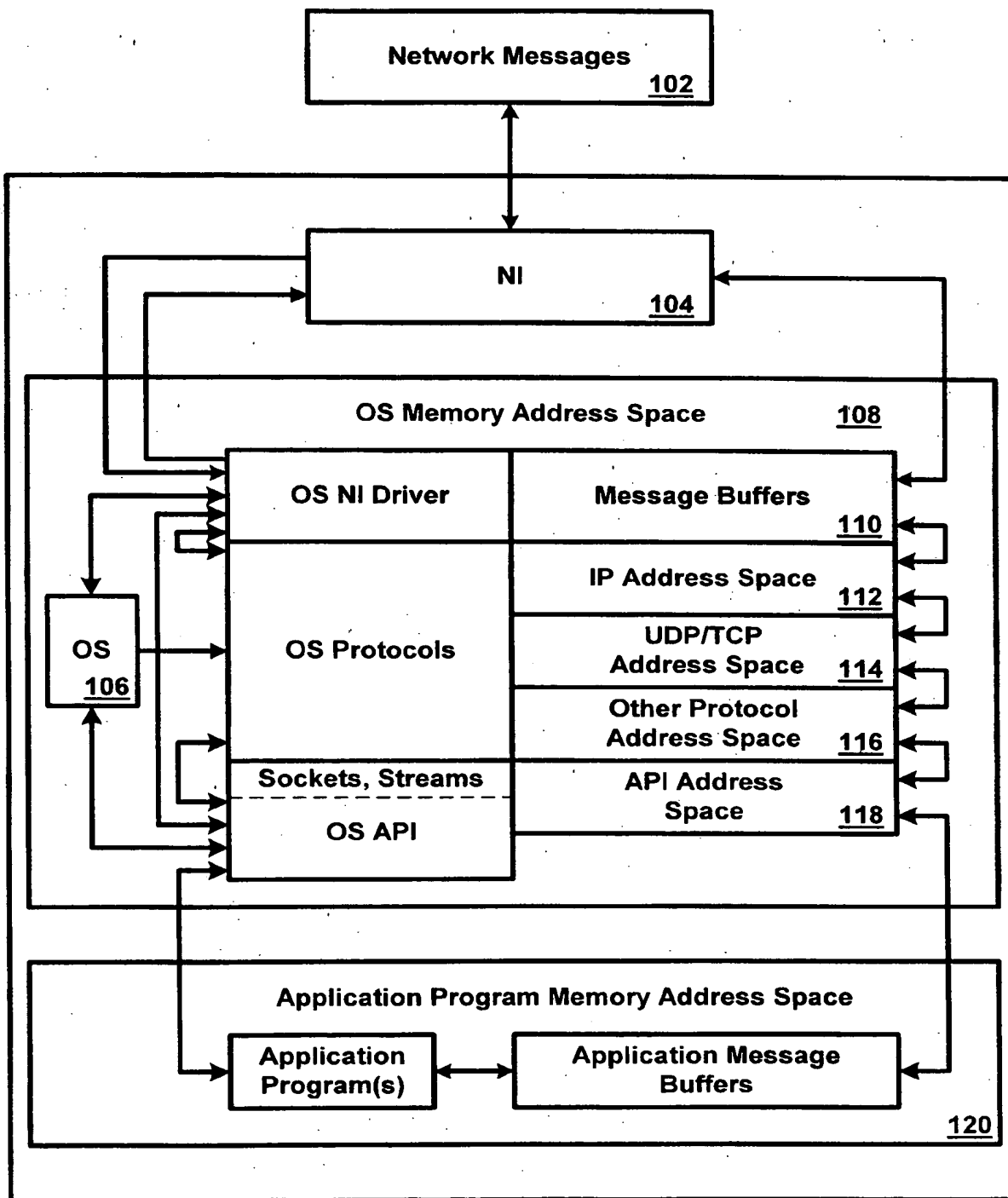


FIGURE 1

2 / 11

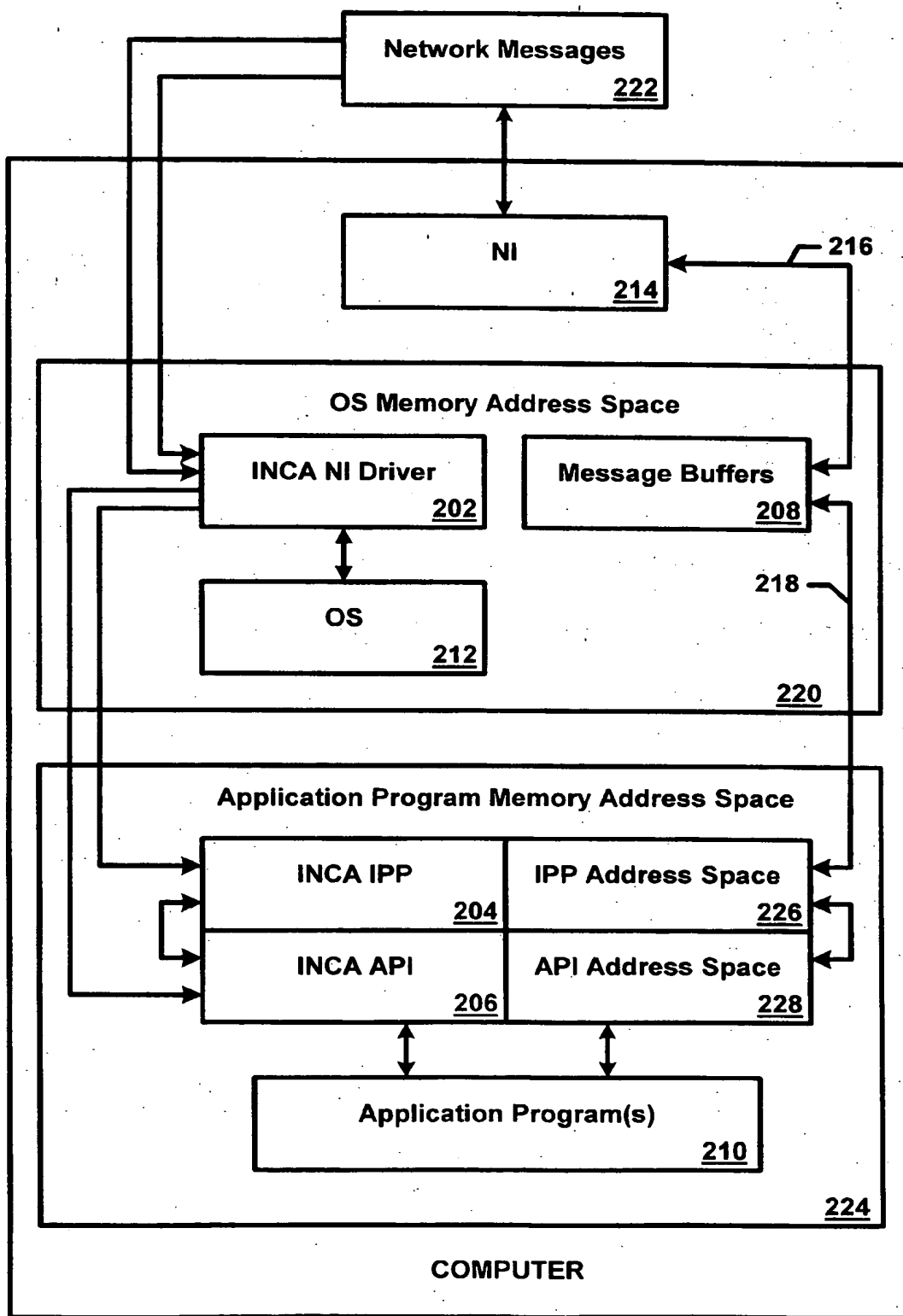


FIGURE 2

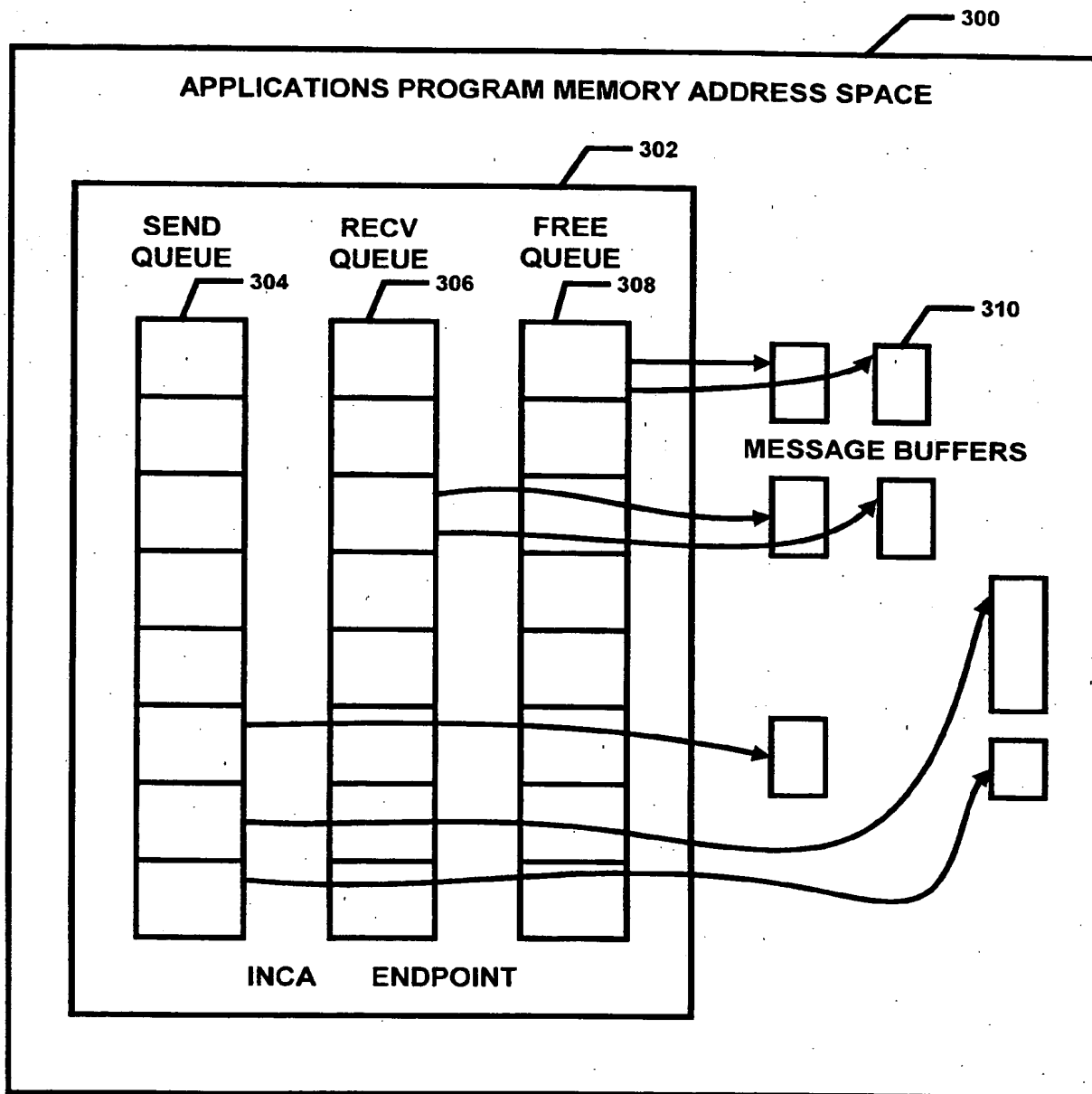


FIGURE 3

4 / 11

```
for( i = 0; i < 1000; i++ )  
    msgData[i]++  
for( i = 0; i < 1000; i++ )  
    msgData[i] = ~ msgData[i];
```

/* LOAD, ADD, STORE */
/* LOAD, COMPLEMENT, STORE */

FIGURE 4a

```
for( i = 0; i < 1000; i++){  
    temp = msgData[i];  
    temp++;  
    temp = ~ temp;  
    msgData[i] = temp;  
}
```

/* LOAD */
/* ADD */
/* COMPLEMENT */
/* STORE */

FIGURE 4b

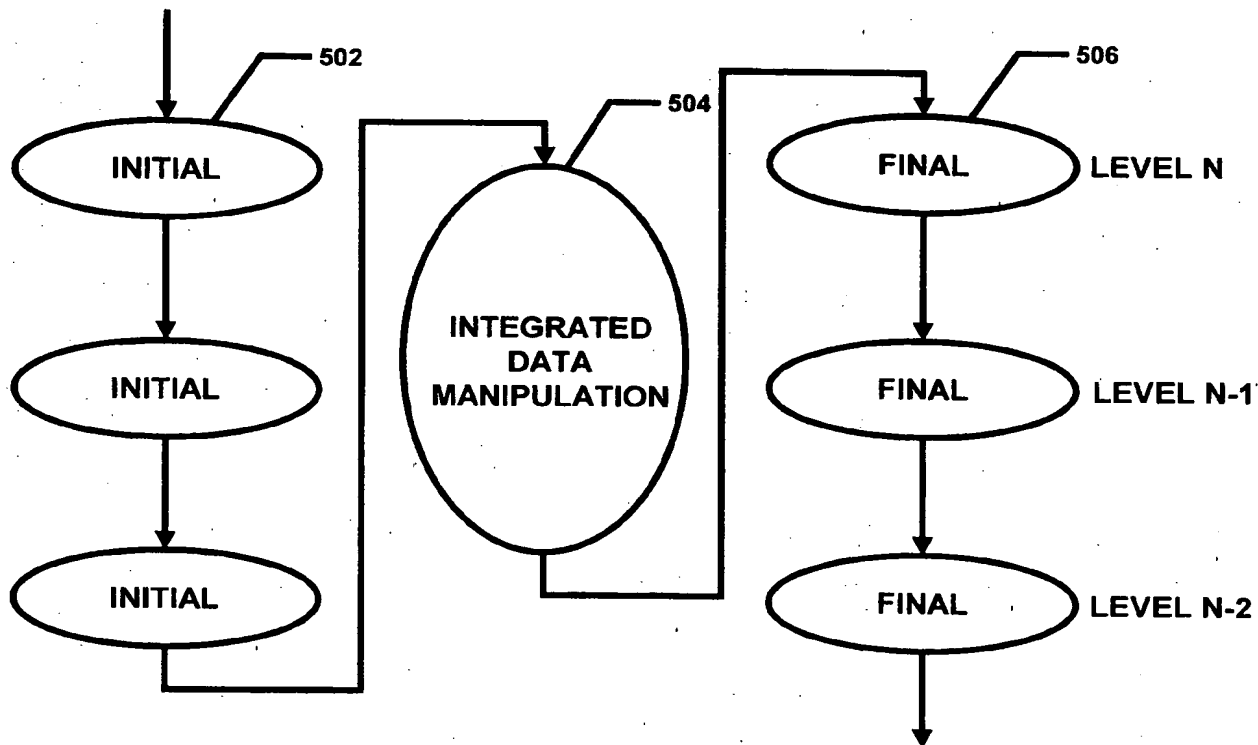


FIGURE 5

6 / 11

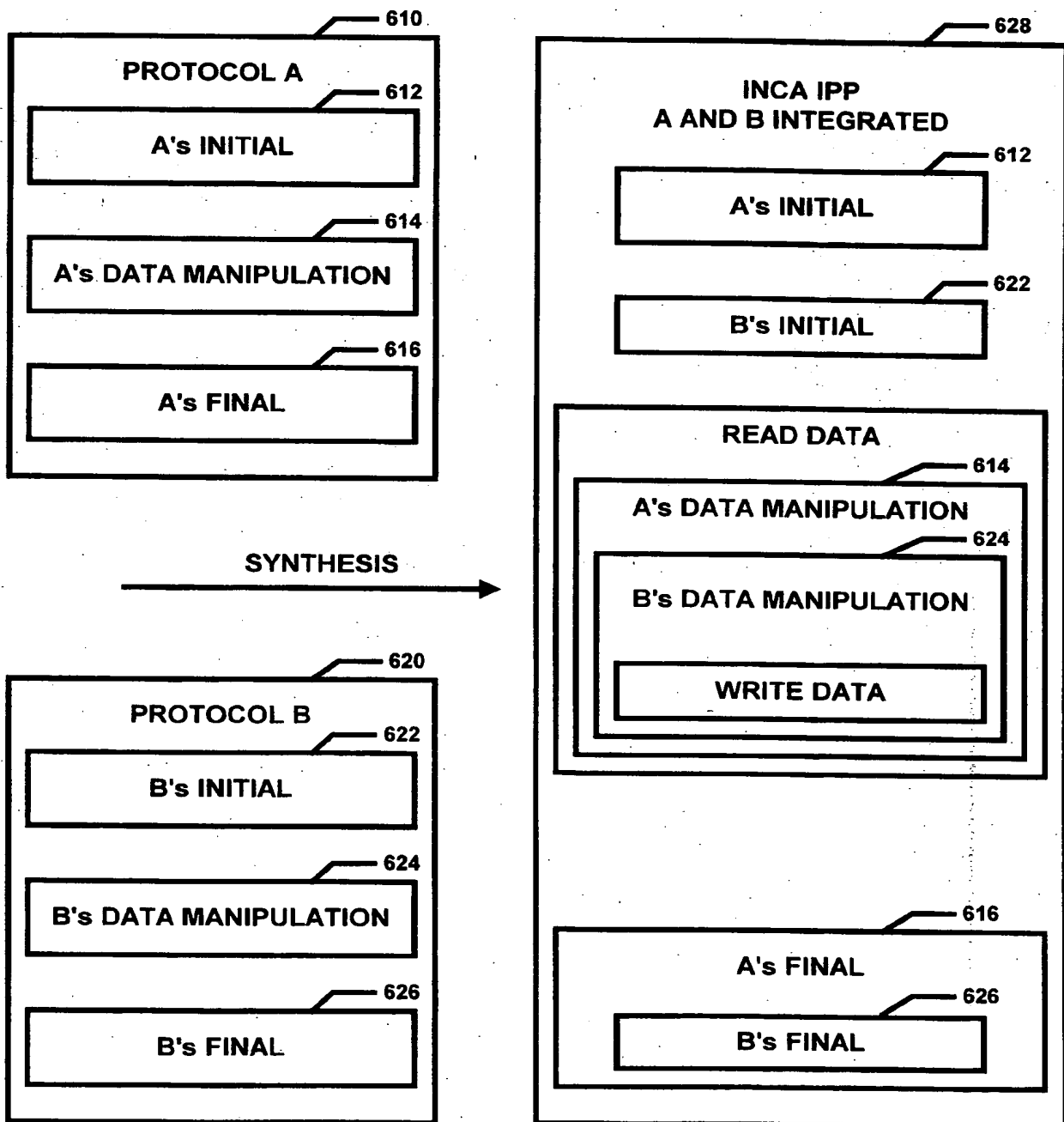


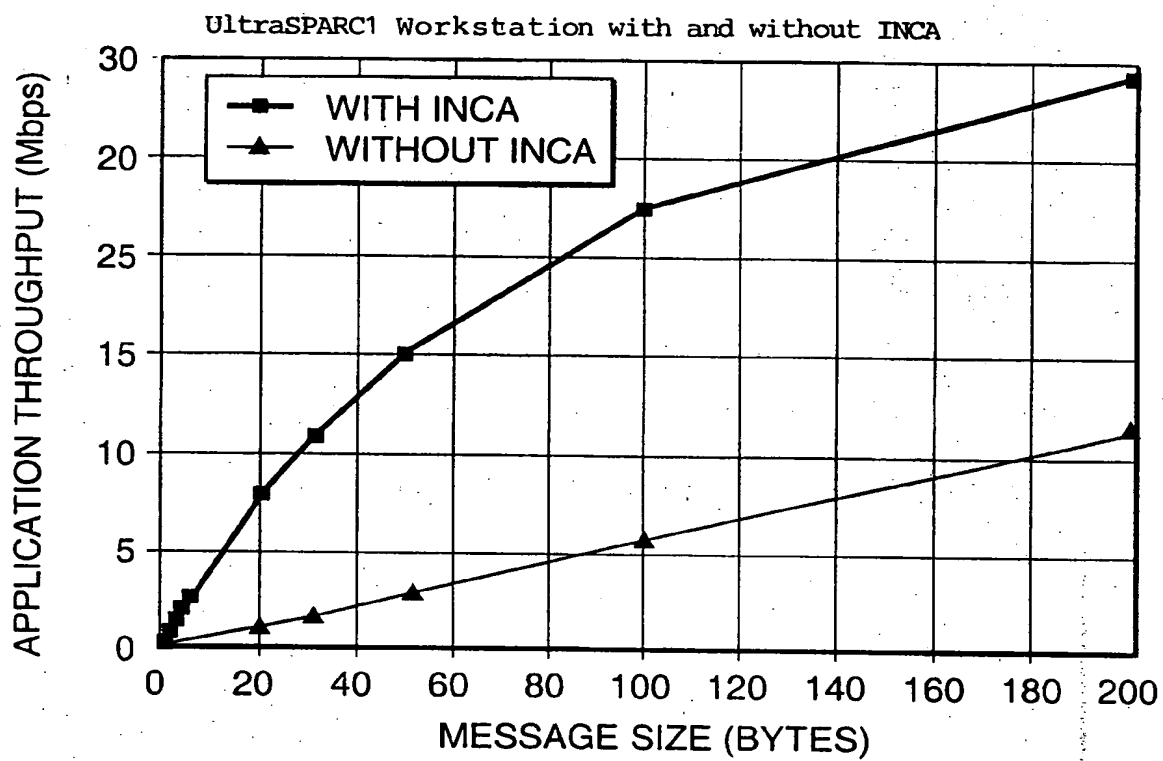
FIGURE 6

- 701. **inca_open()** - open the network device for transmission and set up an OS-Bypass channel for transmission;
- 702. **int inca_open("device name", connector *myconnection)** - myconnection: a pointer to a structure connector which is the data structure the API uses for further processing on the opened device; return value: 0 on success, any other number means an error;
- 703. **inca_close()** - close the device and destroy the OS-Bypass channel;
- 704. **int inca_close("device-name", connector *myconnection)** - return value: 0 on success, any other number means an error;
- 705. **inca_read()** - read data from the network through the OS-Bypass channel created; the read performs all the protocol functions and a pointer to the data buffer is returned;
- 706. **int inca_read(connector *myconnection, int length)** - connector *myconnection: a pointer to the connector buffer returned by **inca_open**; myconnection->inca_recv: points to the data buffer received; int length: number of bytes to be read; return value: returns the number of bytes read;
- 707. **inca_write()** - write data into the network;
- 708. **int inca_write(connector *myconnection, char *buf, int len)** - connector *myconnection: a pointer to the connector buffer returned by **inca_open**; char *buf: a pointer to the data buffer to be transmitted; int length: the number of bytes to be read.

Used mainly by the NI driver but can be used by the application:

- 709. **inca-opendev()** - open the INCA device for use;
- 710. **inca_closedev()** - close the device for use;
- 711. **inca_create_endpoint()** - create an OS-Bypass endpoint;
- 712. **inca_create_chan()** - create a channel for secured multiplexing on the device.

FIGURE 7

**FIG. 8**

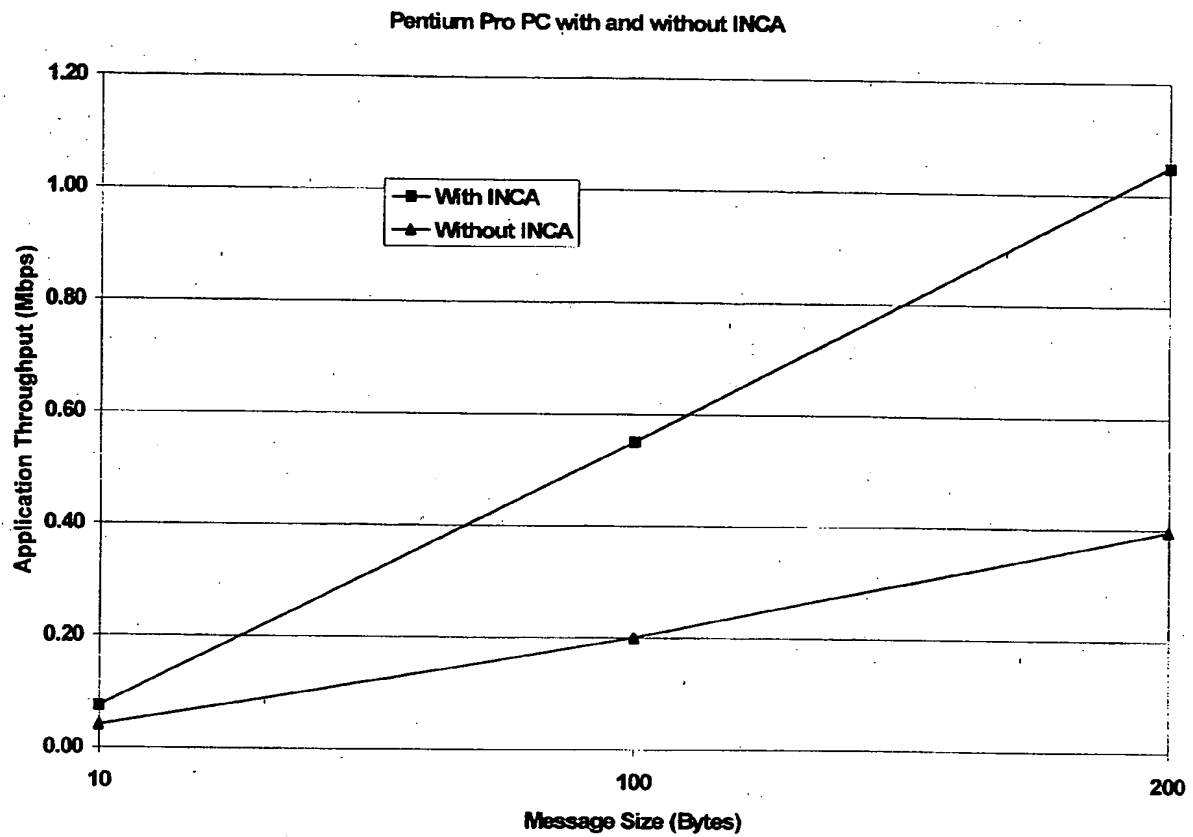


FIGURE 9

10 / 11

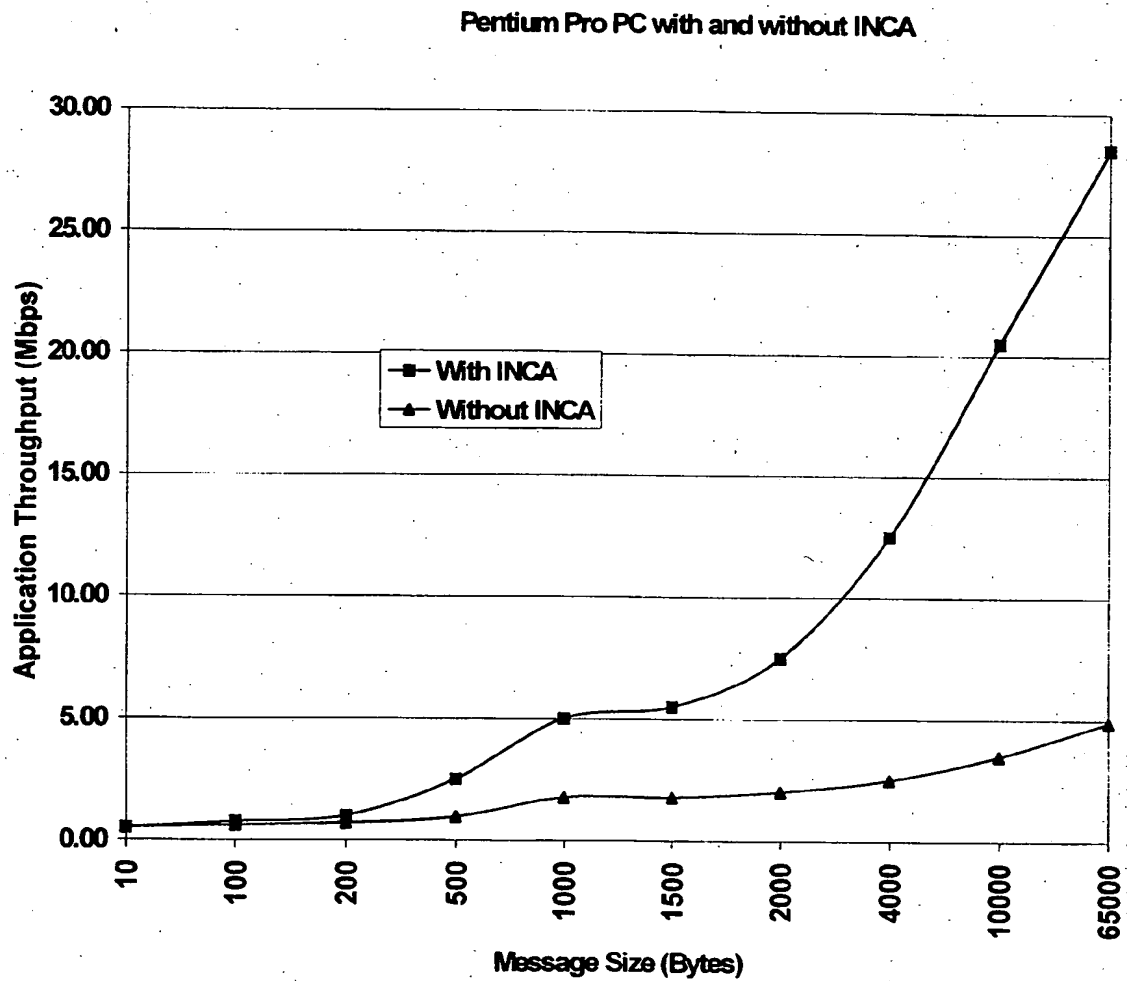


FIGURE 10

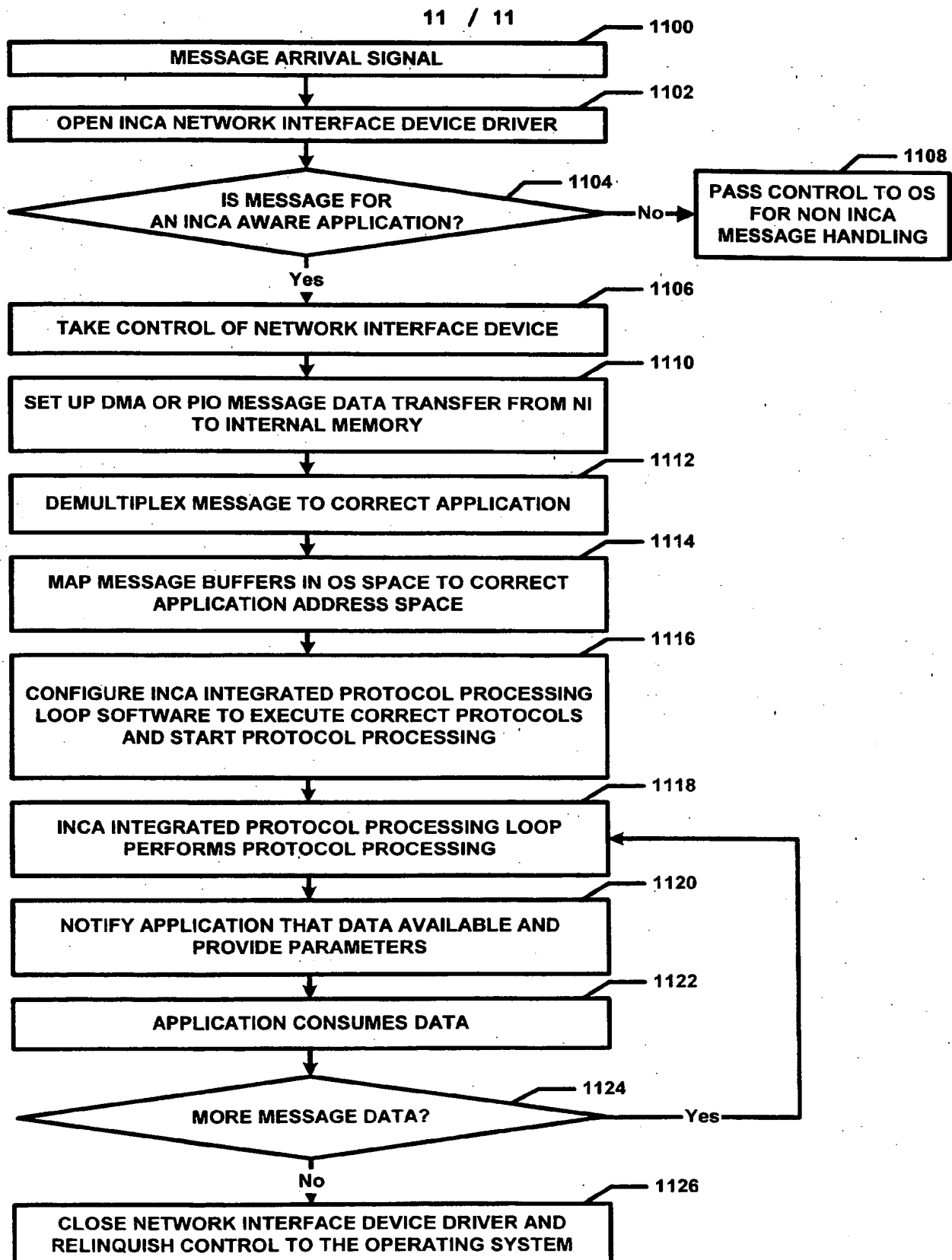


FIGURE 11



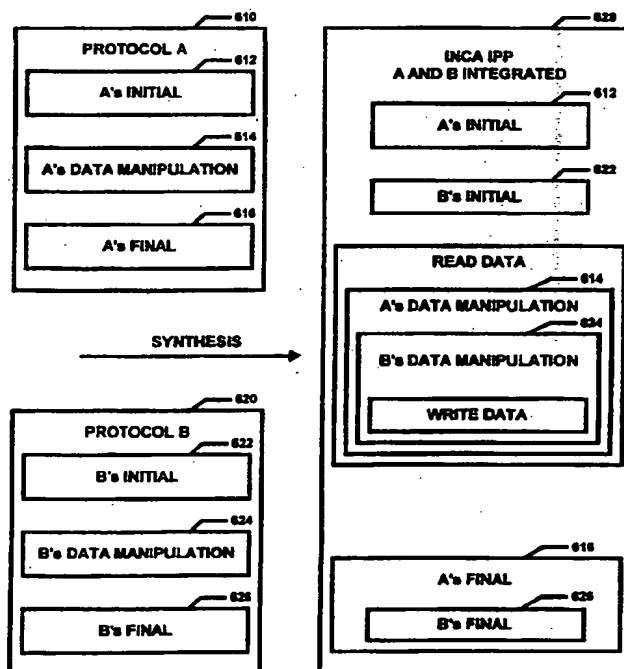
INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification ⁶ : G06F 9/46, H04L 29/06		A3	(11) International Publication Number: WO 99/26377
			(43) International Publication Date: 27 May 1999 (27.05.99)
(21) International Application Number: PCT/US98/24395			
(22) International Filing Date: 16 November 1998 (16.11.98)			
(30) Priority Data: 08/972,157 17 November 1997 (17.11.97) US			
(71) Applicant: MCMZ TECHNOLOGY INNOVATIONS LLC [US/US]; 11029 Grassy Knoll Terrace, Germantown, MD 20876 (US).			
(72) Inventor: SCHUG, Klaus, H.; 3360 Gunnison Drive, Fort Collins, CO 80526-2790 (US).			
(74) Agents: ROBERTS, Jon, L. et al.; Roberts & Brownell, LLC, Suite 212, 8381 Old Courthouse Road, Vienna, VA 22182 (US).			
		(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).	
		Published <i>With international search report.</i> <i>Before the expiration of the time limit for amending the claims</i> <i>and to be republished in the event of the receipt of amendments.</i>	
		(88) Date of publication of the international search report: 16 September 1999 (16.09.99)	

(54) Title: A HIGH PERFORMANCE INTEROPERABLE NETWORK COMMUNICATIONS ARCHITECTURE (INCA)

(57) Abstract

An interoperable, software only network communications architecture (INCA) is presented that improves the internal throughput of network communicated data of workstation and PC class computers at the user level, application program level, by 260 % to 760 %. The architecture is unique because it is interoperable with all existing programs, computers and networks requiring minimal effort to set up and use. INCA operates by mapping network data between the application and operating address space without copying the data, integrating all protocol execution into a single processing loop (628) in the application address space, performing protocol checksumming on a machine word size of data within the protocol execution loop, and providing an application program interface very similar to existing application program interfaces. The network interface driver functions are altered to set up network data transfers to and from the application address space without copying of the data to the OS address space, while buffer management, application to message multiplexing/demultiplexing and security functions are also being performed by the modified network interface driver software. Protocols (610, 620) are executed in the application address space in a single integrated protocol processing loop (628), that interfaces directly to the INCA NI driver on one end and to the application on the other end in order to minimize the amount of times that network communicated data must travel across the internal memory bus. A familiar looking application program interface is provided that differs only slightly from existing application program interfaces which allows existing applications to use the new software with a minimum of effort and cost.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece	ML	Mali	TR	Turkey
BG	Bulgaria	HU	Hungary	MN	Mongolia	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MR	Mauritania	UA	Ukraine
BR	Brazil	IL	Israel	MW	Malawi	UG	Uganda
BY	Belarus	IS	Iceland	MX	Mexico	US	United States of America
CA	Canada	IT	Italy	NE	Niger	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NL	Netherlands	VN	Viet Nam
CG	Congo	KE	Kenya	NO	Norway	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NZ	New Zealand	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	PL	Poland		
CM	Cameroon	KR	Republic of Korea	PT	Portugal		
CN	China	KZ	Kazakstan	RO	Romania		
CU	Cuba	LC	Saint Lucia	RU	Russian Federation		
CZ	Czech Republic	LJ	Liechtenstein	SD	Sudan		
DE	Germany	LK	Sri Lanka	SE	Sweden		
DK	Denmark	LR	Liberia	SG	Singapore		
EE	Estonia						

INTERNATIONAL SEARCH REPORT

International Application No

PCT/US 98/24395

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F9/46 H04L29/06

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

IPC 6 G06F H04L

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
----------	--	-----------------------

X	DRUSCHEL P: "OPERATING SYSTEM SUPPORT FOR HIGH-SPEED COMMUNICATION" COMMUNICATIONS OF THE ASSOCIATION FOR COMPUTING MACHINERY, vol. 39, no. 9, September 1996 (1996-09), pages 41-51, XP000642200	1, 4-10, 14-17, 19
Y	page 42, left-hand column, line 1 - line 5 page 46, right-hand column, line 12 - line 14 page 47, right-hand column, line 11 - line 21 page 47, right-hand column, line 54 - page 48, left-hand column, line 9 page 48, left-hand column, line 52 - line 59 page 48, right-hand column, line 13 - -/--	2, 3, 13, 18, 20, 21

☒ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier document but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance: the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance: the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.
"&" document member of the same patent family

Date of the actual completion of the international search

7 July 1999

Date of mailing of the international search report

02. AUG. 1999

Name and mailing address of the ISA

European Patent Office, P.B. 5818 Patentlaan 2
NL - 2280 HV Rijswijk
Tel. (+31-70) 340-2040, Tx. 31 651 epo nl,
Fax: (+31-70) 340-3016

Authorized officer

Masche, C

INTERNATIONAL SEARCH REPORT

International Application No

PCT/US 98/24395

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
	<p>page 49, left-hand column, line 22 page 49, left-hand column, line 59 - right-hand column, line 26 figure 3</p> <p>----</p>	
Y A	<p>EP 0 707 266 A (SUN MICROSYSTEMS INC) 17 April 1996 (1996-04-17)</p> <p>abstract column 2, line 19 - line 20 page 3, line 47 - page 5, line 9</p> <p>----</p>	<p>2,3</p> <p>1,4-7,9, 10,14</p>
X	<p>EICKEN VON T ET AL: "U-NET: A USER-LEVEL NETWORK INTERFACE FOR PARALLEL AND DISTRIBUTED COMPUTING" OPERATING SYSTEMS REVIEW (SIGOPS), vol. 29, no. 5, 1 December 1995 (1995-12-01), pages 40-53, XP000584816</p>	<p>1,4</p>
Y A	<p>page 43, left-hand column, line 56 - right-hand column, line 4 page 43, right-hand column, line 24 - line 32 page 44, left-hand column, line 13 - line 19 page 44, right-hand column, line 1 - line 33 page 45, left-hand column, line 31 - line 39 page 45, left-hand column, line 56 - line 59 page 45, right-hand column, line 25 - line 39</p> <p>----</p>	<p>13,18 5,6, 8-10,16, 17</p>
X	<p>BURD D: "ZERO-COPY INTERFACING TO TCP/IP" DR. DOBB'S JOURNAL, vol. 20, no. 9, September 1995 (1995-09), pages 68, 70, 72, 74, 76, 78, 106, 108-110, XP000672215</p>	<p>1,4</p>
A	<p>page 68, column 1, line 12 - line 18 page 68, column 3, line 10 - line 42</p> <p>----</p>	<p>2,3,5,6</p>
X A	<p>EP 0 790 564 A (TANDEM COMPUTERS INC) 20 August 1997 (1997-08-20)</p> <p>abstract column 3, line 10 - line 32 column 9, line 26 - column 11, line 29</p> <p>----</p>	<p>1</p> <p>2,4,5,17</p>
	<p>----- -/--</p>	

INTERNATIONAL SEARCH REPORT

Int. l.ional Application No

PCT/US 98/24395

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	<p>NEGISHI Y ET AL: "A PORTABLE COMMUNICATION SYSTEM FOR VIDEO-ON-DEMAND APPLICATIONS USING THE EXISTING INFRASTRUCTURE"</p> <p>PROCEEDINGS OF IEEE INFOCOM 1996. CONFERENCE ON COMPUTER COMMUNICATIONS, FIFTEENTH ANNUAL JOINT CONFERENCE OF THE IEEE COMPUTER AND COMMUNICATIONS SOCIETIES. NETWORKING THE NEXT GENERATION SAN FRANCISCO, MAR. 24 - 28, 1996, vol. 1, no. CONF. 15, 24 March 1996 (1996-03-24), pages 18-26, XP000622290</p> <p>INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS</p>	22,23
Y		20,21,
A	<p>page 20, right-hand column, line 10 -</p> <p>page 21, left-hand column, line 7</p> <p>page 25, right-hand column, line 6 - line 12</p> <p>---</p>	26-28 1,2,7,29
X	<p>O'BRYAN J ET AL: "XNS - X.25 COMMUNICATIONS GATEWAY"</p> <p>21ST. CENTURY MILITARY COMMUNICATIONS - WHAT'S POSSIBLE ?, SAN DIEGO, OCT. 23 - 26, 1988,</p> <p>vol. 3, 23 October 1988 (1988-10-23), pages 1057-1061, XP000011148</p> <p>INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS</p>	22,23
A	<p>page 1058, right-hand column, line 25 - line 33</p> <p>page 1059, left-hand column, line 1 - line 22</p> <p>page 1059, right-hand column, line 45 - line 50</p> <p>---</p>	1,20,21, 29
Y	<p>"TRANSMISSION CONTROL PROTOCOL/INTERNET PROTOCOL CHECKSUM IMPROVEMENT FOR AIX3.2 UNDER RISC SYSTEM/6000"</p> <p>IBM TECHNICAL DISCLOSURE BULLETIN, vol. 37, no. 2A,</p> <p>1 February 1994 (1994-02-01), pages 253-256, XP000432638</p> <p>ISSN: 0018-8689</p> <p>page 255, line 11 - line 22</p> <p>---</p> <p>-/--</p>	26-28

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 98/24395

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 5 638 370 A (LEWIS GLENN ET AL) 10 June 1997 (1997-06-10) abstract column 3, line 41 - line 44 column 9, line 16 - line 25 column 14, line 49 - line 65 -----	26-28
P,X	US 5 701 316 A (MCBREEN JAMES R ET AL) 23 December 1997 (1997-12-23) abstract column 6, line 45 - line 52 column 7, line 1 - line 25 -----	26-28

INTERNATIONAL SEARCH REPORT

International application No.

PCT/US 98/24395

Box I Observations where certain claims were found unsearchable (Continuation of Item 1 of first sheet)

This International Search Report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:
2. ☐ Claims Nos.:
because they relate to parts of the International Application that do not comply with the prescribed requirements to such an extent that no meaningful International Search can be carried out, specifically:
3. ☐ Claims Nos.:
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box II Observations where unity of invention is lacking (Continuation of Item 2 of first sheet)

This International Searching Authority found multiple inventions in this international application, as follows:

SEE ADDITIONAL SHEET

1. ☒ As all required additional search fees were timely paid by the applicant, this International Search Report covers all searchable claims.
2. ☐ As all searchable claims could be searched without effort justifying an additional fee, this Authority did not invite payment of any additional fee.
3. ☐ As only some of the required additional search fees were timely paid by the applicant, this International Search Report covers only those claims for which fees were paid, specifically claims Nos.:
4. ☐ No required additional search fees were timely paid by the applicant. Consequently, this International Search Report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:

Remark on Protest

- ☐ The additional search fees were accompanied by the applicant's protest.
- ☒ No protest accompanied the payment of additional search fees.

INTERNATIONAL SEARCH REPORT

International Application No. PCT/US 98/24395

FURTHER INFORMATION CONTINUED FROM PCT/ISA/ 210

This International Searching Authority found multiple (groups of) inventions in this international application, as follows:

1. Claims: 1-19, 29-33

Method to transfer network communicated data directly from network interface to application address space with only one physical copy of these data

2. Claims: 20-25

Integrating separate protocol processing loops

3. Claims: 26-28

Calculating protocol checksums one word at a time

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No

PCT/US 98/24395

Patent document cited in search report		Publication date	Patent family member(s)	Publication date
EP 0707266	A	17-04-1996	JP 9026929 A	28-01-1997
EP 0790564	A	20-08-1997	CA 2193343 A	21-06-1997
			JP 9325944 A	16-12-1997
US 5638370	A	10-06-1997	NONE	
US 5701316	A	23-12-1997	NONE	

**This Page is Inserted by IFW Indexing and Scanning
Operations and is not part of the Official Record**

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☒ **BLACK BORDERS**
- ☐ **IMAGE CUT OFF AT TOP, BOTTOM OR SIDES**
- ☐ **FADED TEXT OR DRAWING**
- ☐ **BLURRED OR ILLEGIBLE TEXT OR DRAWING**
- ☐ **SKEWED/SLANTED IMAGES**
- ☐ **COLOR OR BLACK AND WHITE PHOTOGRAPHS**
- ☐ **GRAY SCALE DOCUMENTS**
- ☐ **LINES OR MARKS ON ORIGINAL DOCUMENT**
- ☐ **REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY**
- ☐ **OTHER:** _____

IMAGES ARE BEST AVAILABLE COPY.

As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.